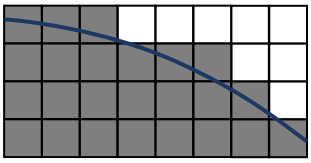
A novel technique to draw antialiased circles without floating point math nor square root

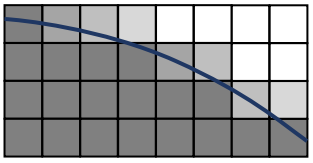
Juan Ramón Vadillo (Versa Design S.L.) – May 28th, 2023

Many resource-limited processors such as microcontrollers, do not have any floating-point processing unit and square root calculation is done by fixed point iterative algorithms that consume a large amount of computing power. However. they are still used for controlling small displays with limited resolutions where antialiasing techniques can provide better visual renderings.



Non-antialiased circle

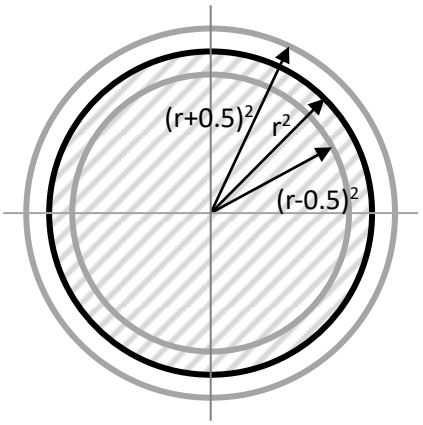
The typical non-antialiased filled circle algorithm removes the need of calculating the distance to the center of the circle by comparing squared distance to squared radius.



Antialiased circle

Squared radius is normally precomputed and a simple comparison x2+y2 < r2 is used to determine whether a pixel is inside the circle or not.

If we want to drawn an antialiased edge, we need to variate linearly the intensity of the pixels in the edge between the radius minus a half of pixel distance and the radius plus a half of pixel distance.



Since we have to run our algorithm without using floating point arithmetic, we can pre-calculate the inner and outer half pixel squared radius as follows:

$$(r_{inner})^2 = (r - 0.5)^2 = r^2 - 2 \cdot r \cdot 0.5 + 0.25 \cong r^2 - r$$

$$(r_{outer})^2 = (r + 0.5)^2 = r^2 + 2 \cdot r \cdot 0.5 + 0.25 \cong r^2 + r$$

To accelerate the algorithm, we need to keep doing the comparison of the squared distance to the center of the circle for each pixel in the circle bounding box:

$$d^2 = squared\ distance\ to\ center = (x^2 + y^2)$$

If $d^2 < r^2$ then the pixel is solid and the intensity is 1. But if the pixel lies between the inner and outer squared radius of the r ± half-pixel circles, then the intensity of the pixel intensity can be interpolated linearly between the inner and outer squared radius.

$$Pixel\ Intensity(d^2) = \frac{(r_{outer})^2 - d^2}{(r_{outer})^2 - (r_{inner})^2}$$

We can also pre-calculate the denominator of this fraction to improve algorithm speed:

$$(r_{outer})^2 - (r_{inner})^2 = (r + 0.5)^2 - (r - 0.5)^2 = 2 \cdot r$$

If the transparency channel can range from 0 to 255, then the resulting code will be:

```
void GrDrawFilledCircleAntialiased(uint16_t x0, uint16_t y0, uint16_t r)
{
    uint32_t x, y, rmin, rmax, sqy, sqd, c;
    rmin = r * r - r;
    rmax = r * r + r;
    for (y = y0 - r; y <= y0 + r; y++)
    {
        sqy = (y - y0) * (y - y0);
        for (x = x0 - r; x <= x0 + r; x++)
        {
            sqd = (x - x0) * (x - x0) + sqy;
            if (sqd < rmin)
            {
                PutPixel (x, y, 255);
            }
            else if (sqd < rmax)
            {
                c = rmax - sqd;
                c *= 256;
                c /= 2 * r;
                if (c > 255) c = 255;
                PutPixel(x, y, c);
            }
        }
    }
}
```

If additional speed improvement is needed, a four-quadrant drawing strategy with a for-loop premature break can be followed.

This novel technique can do the complete filled circle computation without using floating point arithmetic nor square roots, yet providing a very accurate and smoothed edge visual representation. This algorithm can be used in processors with very limited resources, since only involves multiplications, additions and divisions of integer numbers.